CCNU at SemEval-2025 Task 8: Enhancing Question Answering on Tabular Data with Two-Stage Corrections

Chenlian Zhou[1,2,3], **Xilu Cai**[1,2,4], **Yajuan Tong**[1,2,4], **Chengzhao Wu**[1,2,4], **Xin Xu**[1,2,4], **Guanyi Chen**[1,2,4*], and **Tingting He**[1,2,4*]

[1]Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning
[2]National Language Resources Monitor and Research Center for Network Media
[3]Faculty of Artificial Intelligence in Education
[4]School of Computer Science, Central China Normal University
{g.chen, tthe}@ccnu.edu.cn
{myphyllis, cxilu03, tongauan, wcz, xinxu}@mails.ccnu.edu.cn

Abstract

We present the system developed by the Central China Normal University (CCNU) team for the SemEval-2025 shared task 8, which focuses on Question-Answering (QA) for tabular data. Our approach leverages multiple Large Language Models (LLMs), conducting tabular QA as code completion. Additionally, to improve its reliability, we introduce a two-stage corrections mechanism, in which we instruct the LLM to correct the code according to the judges of whether the code is executable and whether the answer obtained from executing the code is semantically consistent with the question. The experiment demonstrates that code correction works but answer correction does not. Finally, we discuss other unsuccessful approaches explored during our development process.

1 Introduction

Large language models (LLMs) have demonstrated strong performance in question answering (QA) tasks (Kamalloo et al., 2023; Mao et al., 2024), including tabular QA (Chen, 2023), where inputs consist of non-database tables. To systematically evaluate the performance of LLMs on tabular QA, Grijalba et al. (2024) introduced DataBench, a benchmark comprising a diverse collection of tabular datasets spanning various domains and question types.

Unfortunately, as noted in the evaluations by Grijalba et al. (2024), LLMs remain unreliable for tabular QA, with substantial room for improvement across all question types and domains. To address this challenge, Os'es Grijalba et al. (2025) organized SemEval-2025 Task 8, based on the DataBench benchmark, to investigate the capability limits of LLMs in tabular QA. This paper presents the solution developed by the Central China Normal University (CCNU) team.

Interestingly, the evaluations by Grijalba et al. (2024) revealed that framing tabular QA as a code completion task can significantly enhance the performance of large language models (LLMs) compared to directly answering questions. Specifically, instead of providing the entire input table, they only exposed the LLM to its meta-information (e.g., column names) and instructed it to generate a Python function that computes the answer to the given question.

As discussed in Grijalba et al. (2024), one limitation of framing QA as code completion is the reliance on a third actor—the Python interpreter—introducing multiple steps in the QA process, each of which can introduce errors. Specifically: (1) The code generated by LLMs may contain errors, leading the Python interpreter to return error messages instead of valid answers. In the evaluations by Grijalba et al. (2024), many incorrect responses were not actually undesired answers but rather error messages. (2) Since LLMs in this paradigm generate code without directly accessing the final outputs their code produces, the answers may be semantically irrelevant to the question. This issue arises because LLMs cannot inherently ensure that the generated answers match the expected type or format of the question.

Our solution builds upon the concept of QA as code completion while specifically addressing the two inherent issues mentioned earlier. To achieve this, we introduce *Two-Stage Corrections*. In the first stage, beyond simply generating code, our approach instructs LLMs to refine their output by incorporating corrections based on error messages from the Python interpreter. In the second stage, LLMs are prompted to verify whether the results obtained from executing the code are semantically consistent with the given questions.

In the following sections, we provide a detailed introduction to the DataBench benchmark (Section 2) and our proposed solution (Section 3). We then

*Corresponding Authors

Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025), pages 841–845

Type	Example
boolean	True/False, Y/N, Yes/No
category	apple
number	10, 20, 30
list[category]	[apple, orange, banana]
list[number]	[1,2,3]

Table 1: Types of QA pairs in DataBench.

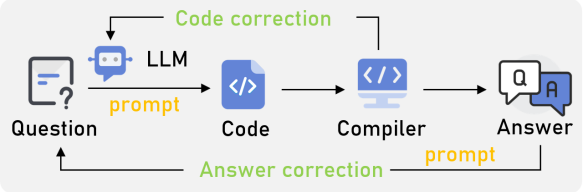

Figure 1: Overview of two-stage corrections for table QA as code completion.

analyze the effectiveness of our Two-Stage Corrections (Section 4). While our results indicate that code correction is successful, answer correction falls short—likely due to the limitations of LLMs in accurately evaluating semantic consistency. Finally, we discuss other unsuccessful approaches we explored during this shared task (Section 5) and reflect on our key findings.

2 Task Description

SemEval-2025 Task 8 is built upon the DataBench benchmark. DataBench consists of 65 tabular datasets from various domains (Grijalba et al., 2024). For each dataset, they hired human participants to write 20 questions, which resulted in 1,300 questions in total. To further improve diversity, it was ensured that the collected questions covered 5 different question types, including *boolean, category, list[category], and list[number]*, as shown in Table 1.

Additionally, Grijalba et al. (2024) also compiled a reduced version of DataBench with an aim of evaluating LLMs that are unable to process large tables, namely, DataBench Lite. Specifically, for each table in each dataset in DataBench, DataBench Lite uses only the first 20 rows.

3 Methodology

As mentioned earlier, we follow Grijalba et al. (2024) in formulating tabular QA as code completion and introduce a Two-Stage Corrections mechanism to address errors arising in the two key stages of this approach: code completion and code execution. This mechanism consists of code correction and answer correction, which aim to improve the reliability of generated solutions. An overview of our method is illustrated in Figure 1. In this section, we first describe how we incorporate the approach of Grijalba et al. (2024) into our solution for tabular QA, followed by a detailed explanation of the Two-Stage Corrections mechanism.

Previous error: {error_message} **Please correct the following code**: {code}

Table 2: Prompt for Code Correction

3.1 Question Answering as Code Completion

Following Grijalba et al. (2024), we came up with the prompt shown in Listing 1 for Tabular QA as code completion.

Listing 1: Prompt for QA as Code Completion

```
# Task Description:
# 1. Determine the type of the answer,
    which should belong to one of the
    following five types: boolean,
    category, number, list[category],
    list[number].
# 2. After determining the type,
    complete the following function in
    one line to answer the question.
# 3. Please specify the type as a
    comment before the code, for example
    : # Type: number

# Question: {question}
# Dataset columns: {columns}
def answer(df: pd.DataFrame):
    df.columns = {columns}
    return
```

The prompt begins with several lines of comments that define the task the LLM needs to accomplish. Specifically, it instructs the LLM to generate a single line of code based on the given question and the meta-information of the table. To enhance the model's awareness of the question type, we introduce a modification to the approach of Grijalba et al. (2024) by adding an extra comment line that asks the LLM to explicitly specify the question type as a line of comment before completing the code. The prompt concludes with the beginning of a function definition, leaving the final line for the LLM to complete. Once generated, the completed code is executed in a Python interpreter to produce the final answer.

Main Task: As a professional data analyst, your task is to determine if the generated answer is correct based on the provided tabular knowledge and question. Correctness is defined as: the answer is completely consistent with the tabular knowledge and accurately answers the question.
Judgment Criteria:

1. If the generated answer is completely consistent with the factual content in the tabular data and directly answers the question, it is judged as "Correct".
2. If the generated answer is inconsistent with the tabular data or the requirements of the question, whether it is a partial error or a logical error, it should be judged as "Incorrect".

Reasoning Requirements: As an expert, you need to provide a step-by-step reasoning process, explaining how to extract relevant information from the tabular knowledge and verify the generated answer. The reasoning process should include the following:

- Identify the source of information in the table that is relevant to the question.
- Explain how this information supports or refutes the generated answer.
- Compare the differences between the generated answer and the tabular data, and explain the reasons for the judgment.

Output Format:

- **Correctness Judgment:** [Correct/Incorrect]
- **Reasoning Process:** Detailed reasoning process, explaining whether the answer is correct and the reasons for it.

Example:
Input:
{example}
Current Input:
Tabular Knowledge: {table_knowledge}
Question: {question}
Generated Answer: {generated_answer}

Table 3: Prompt for judging the correctness of an answer.

3.2 Two-Stage Corrections

After obtaining the code, we perform our two-stage corrections: before and after the Python interpreter successfully generates the final output.

3.2.1 Code Correction

If the code contains errors, which often occur due to the LLM's misinterpretation of the tabular structure, the interpreter returns an error message instead of a valid output. To correct the code, we prompt the LLM to revise its own code based on the error message, using the instruction provided in Table 2.

3.2.2 Answer Correction

To verify whether the answers generated from successfully executed code are semantically consistent with the given questions, we employ two strategies: (1) Direct Answer Evaluation: We instruct another LLM to assess whether the answer is correct or incorrect based on the given question and table, using the prompt in Table 3. Instead of providing a simple yes or no, the LLM is also required to explain the reasoning behind its judgment. (2) Answer Type Consistency Check: Since determining the correctness of an answer may be too challenging for LLMs, an alternative approach is to evaluate whether the type of the generated answer aligns with the expected question type, using the prompt in Table 4. Finally, the LLM receives both judgment and explanation, which it then uses to refine and regenerate the code accordingly.

Analyze the following question and determine the expected answer type:

Question: {question}
Possible types:

- *boolean*: yes/no questions.
- *number*: questions requiring numeric answers.
- *list[number]*: questions requiring a list of numbers.
- *list[category]*: questions requiring a list of categories, where categories can include date (e.g., 1970-01-01), text, or url.
- *category*: questions requiring a single category, which can be a date, text, url, or other categorical value.

Return only the type name, nothing else.

Table 4: Prompt for judging the correctness of an answer's type.

4 Experiments

In this section, we start with introducing the backbone LLMs we used in our experiments and report the experimental results.

4.1 Backbone LLMs.

In our experiments, we tried a range of open-sourced LLMs as backbone models. Specifically, the models included Code Llama (Rozière et al., 2023), Qwen2.5-72B-Instruct (Hui et al., 2024), Llama-3-1-8B (Touvron et al., 2023), DeepSeek-Coder-6.7B-Instruct (Guo et al., 2024), DeepSeek-V3 (DeepSeek-AI, 2024), Qwen2.5-72B-Instruct (Team, 2024), and DeepSeek-

Model	w/o CC	w/ CC
Code Llama	20.31	19.38
Qwen2.5-Coder-7B	67.50	68.44
Llama-3-1-8B	66.25	64.69
DeepSeek-Coder-6.7B	72.19	74.38
DeepSeek-V3	80.31	83.44
Qwen2.5-72B	82.81	85.00
DeepSeek-R1 (w/o Think)	79.69	85.00

Table 5: Results of different backbone LLMs with and without Code Correction (CC) on the development set of DataBench.

Model	w/o CC	w/ CC
Code Llama	19.68	17.81
Qwen2.5-Coder-7B	67.19	69.69
Llama-3-1-8B	61.88	64.06
DeepSeek-Coder-6.7B	71.88	72.50
DeepSeek-V3	80.62	82.19
Qwen2.5-72B	81.56	85.31
DeepSeek-R1 (w/o Think)	72.81	85.72

Table 6: Results of different backbone LLMs with and without Code Correction (CC) on the development set of DataBench Lite.

R1 (DeepSeek-AI, 2025). For DeepSeek-R1, we did not enforce it to "think", which may limit its reasoning ability (making it often work in the same way as DeepSeek-V3) but makes its outputs more straightforward and easier to use in subsequent processing steps.

4.2 The Effect of Code Correction

Table 3 and Table 4 present the performance of various LLMs, with and without code correction, on DataBench and DataBench Lite, respectively. The results indicate that code correction improves the performance of nearly all backbone LLMs across both datasets, demonstrating its effectiveness.

When comparing different backbone LLMs, we find that DeepSeek-R1 (without Think) with code correction achieves the highest performance on both datasets.

4.3 The Effect of Answer Correction

To assess the effectiveness of answer correction, we evaluated the accuracy of two judgment tasks: determining the correctness of answers using the prompt in Table 3 and verifying the correctness of answer types using the prompt in Table 4. An experiment using LLaMA-3.1-8B-Instruct as the judge yielded accuracies of 31.10% for answer correctness and 91.38% for answer type consistency. However, subsequent attempts to integrate this judgment mechanism into our system proved unsuccessful, as it either decreased overall performance or had no measurable impact.

4.4 Final Solution

In summary, our final solution adopts DeepSeek-R1 (without Think) as the backbone LLM, utilizing only code correction to enhance performance.

5 Other Unsuccessful Attempts

We also experimented with an ensemble solution, where multiple tabular QA systems generated answers in parallel, followed by a majority vote to determine the final response. However, this approach underperformed compared to simply using DeepSeek-R1 (without Think) with code correction, suggesting that accuracy may lie in the hands of a select few models rather than a collective consensus.

6 Conclusion

This paper presents the Central China Normal University (CCNU) team's solution to SemEval-2025 Task 8, the DataBench task, which requires systems to perform tabular QA. Building on the idea of tabular QA as code completion, we further propose a two-stage correction mechanism comprising code correction and answer correction to enhance reliability. Experimental results show that code correction effectively improves performance, while answer correction does not.

References

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1120–1130, Dubrovnik, Croatia. Association for Computational Linguistics.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *CoRR*, abs/2412.19437.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Jorge Osés Grijalba, Luis Alfonso Ureña López, Eugenio Martínez Cámara, and José Camacho-Collados. 2024. Question answering over tabular data with databench: A large-scale empirical evaluation of llms. In *LREC/COLING*, pages 13471–13488. ELRA and ICCL.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi,

Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *CoRR*, abs/2401.14196.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-coder technical report. *CoRR*, abs/2409.12186.

Ehsan Kamalloo, Nouha Dziri, Charles Clarke, and Davood Rafiei. 2023. Evaluating open-domain question answering in the era of large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5591–5606, Toronto, Canada. Association for Computational Linguistics.

Rui Mao, Guanyi Chen, Xulang Zhang, Frank Guerin, and Erik Cambria. 2024. GPTEval: A survey on assessments of ChatGPT and GPT-4. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 7844–7866, Torino, Italia. ELRA and ICCL.

Jorge Os'es Grijalba, Luis Alfonso Ure na-L'opez, Eugenio Mart'inez C'amara, and Jose Camacho-Collados. 2025. SemEval-2025 task 8: Question answering over tabular data. In *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, Vienna, Austria. Association for Computational Linguistics.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.